```
1  /*
2   * linux/kernel/math/convert.c
3   *
4   * (C) 1991 Linus Torvalds
5   */
6
7  #include <linux/math_emu.h> // 协处理器头文件。定义临时实数结构和 387 寄存器操作宏等。
8
9  /*
10  * NOTE!!! There is some "non-obvious" optimisations in the temp_to_long
11  * and temp_to_short conversion routines: don't touch them if you don't
12  * know what's going on. They are the adding of one in the rounding: the
13  * overflow bit is also used for adding one into the exponent. Thus it
14  * looks like the overflow would be incorrectly handled, but due to the
15  * way the IEEE numbers work, things are correct.
16  *
17  * There is no checking for total overflow in the conversions, though (ie
18  * if the temp-real number simply won't fit in a short- or long-real.)
19  */
    /*
     * 注意!!! 在 temp_to_long 和 temp_to_short 数据类型转换子程序中有些“不明显”
     * 的优化处理：如果不理解就不要随意修改。它们是舍入操作中的加 1：溢出位也同
     * 样被用于向指数中加 1。因此看上去溢出好像没有被正确地处理，但是由于 IEEE
     * 浮点数标准所规定数据格式的操作方式，这些做法是正确的。
     *
     * 不过这里没有对转换过程中总体溢出作检测（也即临时实数是否能够简单地放入短
     * 实数或长实数格式中）。
     */
20
    // 短实数转换成临时实数格式。
    // 短实数长度是 32 位，其有效数（尾数）长度是 23 位，指数是 8 位，还有 1 个符号位。
21  void short_to_temp(const short_real * a, temp_real * b)
22  {
    // 首先处理被转换的短实数是 0 的情况。若为 0，则设置对应临时实数 b 的有效数为 0。然后根
    // 据短实数符号位设置临时实数的符号位，即 exponent 的最高有效位。
23          if (!(*a & 0x7fffffff)) {
24                  b->a = b->b = 0;                        // 置临时实数的有效数 = 0。
25                  if (*a)
26                          b->exponent = 0x8000;           // 设置符号位。
27                  else
28                          b->exponent = 0;
29                  return;
30          }
    // 对于一般短实数，先确定对应临时实数的指数值。这里需要用到整型数偏置表示方法的概念。
    // 短实数指数的偏置量是 127，而临时实数指数的偏置量是 16383。因此在取出短实数中指数值
    // 后需要变更其中的偏置量为 16383。 此时就形成了临时实数格式的指数值 exponent。另外，
    // 如果短实数是负数，则需要设置临时实数的符号位（位 79）。下一步设置尾数值。方法是把
    // 短实数左移 8 位，让 23 位尾数最高有效位处于临时实数的位 62 处。而临时实数尾数的位 63
    // 处需要恒置一个 1，即需要或上 0x80000000。最后清掉临时实数低 32 位有效数。
31          b->exponent = ((*a>>23) & 0xff)-127+16383;      // 取出短实数指数位，更换偏置量。
32          if (*a<0)
33                  b->exponent |= 0x8000;                  // 若为负数则设置符号位。
```

```
34          b->b = (*a<<8) | 0x80000000;                    // 放置尾数，添加固定 1 值。
35          b->a = 0;
36  }
37
    // 长实数转换成临时实数格式。
    // 方法与 short_to_temp()完全一样。不过长实数指数偏置量是 1023。
38  void long_to_temp(const long_real * a, temp_real * b)
39  {
40          if (!a->a && !(a->b & 0x7fffffff)) {
41                  b->a = b->b = 0;                        // 置临时实数的有效数 = 0。
42                  if (a->b)
43                          b->exponent = 0x8000;           // 设置符号位。
44                  else
45                          b->exponent = 0;
46                  return;
47          }
48          b->exponent = ((a->b >> 20) & 0x7ff)-1023+16383;  // 取长实数指数，更换偏置量。
49          if (a->b<0)
50                  b->exponent |= 0x8000;                  // 若为负数则设置符号位。
51          b->b = 0x80000000 | (a->b<<11) | (((unsigned long)a->a)>>21); // 放置尾数，添 1。
52          b->a = a->a<<11;
53  }
54
    // 临时实数转换成短实数格式。
    // 过程与 short_to_temp()相反，但需要处理精度和舍入问题。
55  void temp_to_short(const temp_real * a, short_real * b)
56  {
    // 如果指数部分为 0，则根据有无符号位设置短实数为-0 或 0。
57          if (!(a->exponent & 0x7fff)) {
58                  *b = (a->exponent)?0x80000000:0;
59                  return;
60          }
    // 先处理指数部分。即更换临时实数指数偏置量（16383）为短实数的偏置量 127。
61          *b = ((((long) a->exponent)-16383+127) << 23) & 0x7f800000;
62          if (a->exponent < 0)                            // 若是负数则设置符号位。
63                  *b |= 0x80000000;
64          *b |= (a->b >> 8) & 0x007fffff;                 // 取临时实数有效数高 23 位。
    // 根据控制字中的舍入设置执行舍入操作。
65          switch (ROUNDING) {
66                  case ROUND_NEAREST:
67                          if ((a->b & 0xff) > 0x80)
68                                  ++*b;
69                          break;
70                  case ROUND_DOWN:
71                          if ((a->exponent & 0x8000) && (a->b & 0xff))
72                                  ++*b;
73                          break;
74                  case ROUND_UP:
75                          if (!(a->exponent & 0x8000) && (a->b & 0xff))
76                                  ++*b;
77                          break;
78          }
79  }
```

```
80
   // 临时实数转换成长实数。
81 void temp_to_long(const temp_real * a, long_real * b)
82 {
83         if (!(a->exponent & 0x7fff)) {
84                 b->a = 0;
85                 b->b = (a->exponent)?0x80000000:0;
86                 return;
87         }
88         b->b = (((0x7fff & (long) a->exponent)-16383+1023) << 20) & 0x7ff00000;
89         if (a->exponent < 0)
90                 b->b |= 0x80000000;
91         b->b |= (a->b >> 11) & 0x000fffff;
92         b->a = a->b << 21;
93         b->a |= (a->a >> 11) & 0x001fffff;
94         switch (ROUNDING) {
95                 case ROUND_NEAREST:
96                         if ((a->a & 0x7ff) > 0x400)
97                                 __asm__("add1 $1,%0 ; adc1 $0,%1"
98                                         :"=r" (b->a),"=r" (b->b)
99                                         :"" (b->a),"1" (b->b));
100                        break;
101                case ROUND_DOWN:
102                        if ((a->exponent & 0x8000) && (a->b & 0xff))
103                                __asm__("add1 $1,%0 ; adc1 $0,%1"
104                                        :"=r" (b->a),"=r" (b->b)
105                                        :"" (b->a),"1" (b->b));
106                        break;
107                case ROUND_UP:
108                        if (!(a->exponent & 0x8000) && (a->b & 0xff))
109                                __asm__("add1 $1,%0 ; adc1 $0,%1"
110                                        :"=r" (b->a),"=r" (b->b)
111                                        :"" (b->a),"1" (b->b));
112                        break;
113        }
114 }
115
   // 临时实数转换成临时整数格式。
   // 临时整数也用 10 字节表示。其中低 8 字节是无符号整数值，高 2 字节表示指数值和符号位。
   // 如果高 2 字节最高有效位为 1，则表示是负数；若位 0，表示是正数。
116 void real_to_int(const temp_real * a, temp_int * b)
117 {
118         int shift =  16383 + 63 - (a->exponent & 0x7fff);
119         unsigned long underflow;
120
121         b->a = b->b = underflow = 0;
122         b->sign = (a->exponent < 0);
123         if (shift < 0) {
124                 set_OE();
125                 return;
126         }
127         if (shift < 32) {
128                 b->b = a->b; b->a = a->a;
```

```
129         } else if (shift < 64) {
130                 b->a = a->b; underflow = a->a;
131                 shift -= 32;
132         } else if (shift < 96) {
133                 underflow = a->b;
134                 shift -= 64;
135         } else
136                 return;
137         __asm__("shrdl %2,%1,%0"
138                 :"=r" (underflow), "=r" (b->a)
139                 :"c" ((char) shift), "" (underflow), "1" (b->a));
140         __asm__("shrdl %2,%1,%0"
141                 :"=r" (b->a), "=r" (b->b)
142                 :"c" ((char) shift), "" (b->a), "1" (b->b));
143         __asm__("shrl %1,%0"
144                 :"=r" (b->b)
145                 :"c" ((char) shift), "" (b->b));
146         switch (ROUNDING) {
147                 case ROUND_NEAREST:
148                         __asm__("addl %4,%5 ; adcl $0,%0 ; adcl $0,%1"
149                                 :"=r" (b->a), "=r" (b->b)
150                                 :"" (b->a), "1" (b->b)
151                                 ,"r" (0x7fffffff + (b->a & 1))
152                                 ,"m" (*&underflow));
153                         break;
154                 case ROUND_UP:
155                         if (!b->sign && underflow)
156                                 __asm__("addl $1,%0 ; adcl $0,%1"
157                                         :"=r" (b->a), "=r" (b->b)
158                                         :"" (b->a), "1" (b->b));
159                         break;
160                 case ROUND_DOWN:
161                         if (b->sign && underflow)
162                                 __asm__("addl $1,%0 ; adcl $0,%1"
163                                         :"=r" (b->a), "=r" (b->b)
164                                         :"" (b->a), "1" (b->b));
165                         break;
166         }
167 }
168
    // 临时整数转换成临时实数格式。
169 void int_to_real(const temp_int * a, temp_real * b)
170 {
    // 由于原值是整数，所以转换成临时实数时指数除了需要加上偏置量 16383 外，还要加上 63。
    // 表示有效数要乘上 2 的 63 次方，即都是整数值。
171         b->a = a->a;
172         b->b = a->b;
173         if (b->a || b->b)
174                 b->exponent = 16383 + 63 + (a->sign? 0x8000:0);
175         else {
176                 b->exponent = 0;
177                 return;
178         }
```

```
    // 对格式转换后的临时实数进行规格化处理，即让有效数最高有效位不是 0。
179         while (b->b >= 0) {
180                 b->exponent--;
181                 __asm__("add1 %0,%0 ; adc1 %1,%1"
182                         :"=r" (b->a),"=r" (b->b)
183                         :"" (b->a),"1" (b->b));
184         }
185 }
186
```